## REST APPLICATION DEVELOPMENT USING SPRING BOOT AND JPA

**Overview:** This presentation explores key annotations provided by Spring Boot to simplify REST application development. It covers core annotations, dependency injection mechanisms, data access annotations, testing annotations, and AOP annotations.

**1. Core Annotations**

* **@SpringBootApplication:** The main entry point of a Spring Boot application. It combines @Configuration, @EnableAutoConfiguration, and @ComponentScan to enable auto-configuration and component scanning.
* **@Controller:** Defines a class as a controller in Spring MVC, handling HTTP requests and returning view names.
* **@RestController:** A specialized version of @Controller specifically designed for RESTful web services. Methods return data directly (e.g., JSON or XML) instead of view names.
* **@Service:** Used to indicate a class is a service in the business logic layer. It’s typically applied to classes that perform business logic operations.
* **@Repository:** Indicates a class is a Data Access Object (DAO) in the persistence layer. It helps translate platform-specific exceptions to Spring’s DataAccessException.
* **@Component:** A generic stereotype annotation for components when a more specific one is not applicable.

**2. Dependency Injection**

* **@Autowired:** Used to automatically inject dependencies into a Spring bean. It allows Spring to resolve and inject collaborating beans.
* **@Qualifier:** Used with @Autowired to specify which bean should be injected when multiple candidates of the same type exist.

**3. Data Access Annotations**

* **@Entity:** Defines a data entity managed by the ORM (Object-Relational Mapping) system. It maps class fields to database tables/columns and provides CRUD operations.
* **@Repository:** Indicates a class is a Spring Data repository, typically a DAO. Used to perform CRUD operations on entities.
* **@Transactional:** Defines the scope of a single database transaction. Applied to methods or classes to indicate that the annotated method or all methods within the class should be wrapped with a transaction.
* **@Query:** Defines a query method using JPQL (Java Persistence Query Language) or native SQL in a Spring Data repository.
* **@Column:** Specifies details of a column in a database table. Used to customize the mapping of a field to a column.

**4. Testing Annotations**

* **@RunWith(SpringRunner.class):** Provides a bridge between JUnit and the Spring TestContext Framework.
* **@SpringBootTest:** Specifies the configuration of the ApplicationContext in a Spring Boot test.
* **@Configuration:** Indicates that a class declares one or more @Bean methods and may be processed by the Spring container.
* **@Value:** Injects values from properties files or other sources into fields.
* **@Conditional:** Conditionally activates a component or configuration class based on the evaluation of a Condition.
* **@Scheduled:** Defines when a method should be invoked based on a fixed rate or cron expression.
* **@EnableAutoConfiguration:** Enables Spring Boot’s auto-configuration mechanism.

**5. AOP (Aspect-Oriented Programming) Annotations**

* **@Aspect:** Identifies a class as an aspect. Contains advice methods and pointcuts.
* **@Before, @After, @Around:** Define advice methods to run before, after, or around a method execution.
* **@Pointcut:** Defines a reusable pointcut expression.

**6. Spring Web Annotations**

* **@RequestBody:** Binds the method parameter to the body of the HTTP request.
* **@ResponseStatus:** Sets the HTTP response status code.
* **@ExceptionHandler:** Handles exceptions in a specific controller or globally.
* **@ModelAttribute:** Adds attributes to the model before the handler method is invoked.